第4章

専用ツールによる本格的シミュレーションを体験する
― ModelSim活用チュートリアル

宮島 健

ここでは，米国Mentor Graphics社のシミュレータ「Model Sim」を活用する方法を解説する．FPGA開発では広く用いられているツールである．やや複雑な機能を持つモジュールを例に，RTLシミュレーションを体験する．本誌付属DVD-ROMには，無償で利用できるModelSim-Altera Web Editionを収録している．（編集部）

本稿では，米国Mentor Graphics社の「ModelSim」というシミュレータを使って，どの開発フローにおいても共通の工程であるRTLシミュレーションを体験することにします．

1．シミュレーションを理解する

ASICやFPGAの一般的な開発フローを**図1**に示します．

● LSI開発ではシミュレーションが不可欠

シミュレーションは，その対象がチップ・レベルかブロック・レベルかに関わらず，重要かつ必須の工程です．RTL（resister transfer level）設計をした後にRTLシミュレーションを実施します．論理合成後にはゲート・レベル・シミュレーション，配置配線後には配置配線による遅延を考慮したタイミング・シミュレーションを実施するという流れになります．

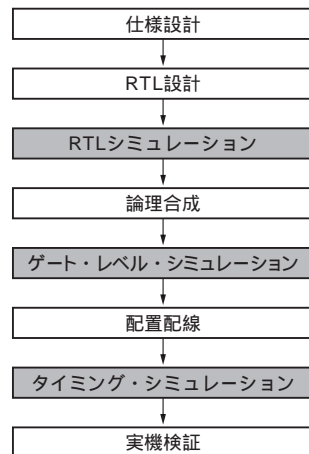
対象デバイスがASICかFPGAかによって異なる点もあります．FPGAの場合，手元で配置配線が行えるため，配置配線後の実遅延情報に基づくタイミング・シミュレーションが可能です．しかしASICの場合は，ASICベンダから提供されている配線容量や階層レベルに基づいた遅延見積もりのプログラムを実行し，配置配線前の仮遅延情報を使ってタイミング・シミュレーションを行うことになります．

また最近では，論理合成後に静的なアプローチ，すなわち等価性検証（formal verification）や静的タイミング解析（STA ：static timing analyze）を実施する傾向にあります．

● RTLシミュレーションと検証

RTLシミュレーションは，設計したRTLコードがどのような振る舞いをするかを知る工程です．設計したRTLコードに対して，それを活性化させるためのパターン（スティミュラスと呼ぶ）を与えることにより行います．

このしくみからわかるように，シミュレータはRTLコー

図1 ASIC/FPGAの一般的な開発フロー
RTL（resister transfer level）設計後にRTLシミュレーション，論理合成後にゲート・レベル・シミュレーション，配置配線後に，配置配線による遅延を考慮したタイミング・シミュレーションを実施する．

KeyWord Model Sim，RTLシミュレーション，ゲート・レベル・シミュレーション，タイミング・シミュレーション，等価性検証，静的タイミング検証，スティミュラス，アサーション，SRAM制御回路，テストベンチ

ドの振る舞いが正しいか正しくないかの判定はしません．振る舞いの正当性を確認する工程は，検証（verification）と呼ばれています．

シミュレーションによる検証では，結果である波形を設計者が目視確認したり，期待される振る舞い（期待値）やリファレンスとなるモデルとの自動比較を行うことで正当性を判断します．最近ではアサーションを埋め込むという方法も用いられるようになってきています．

2．ModelSimによるシミュレーションを体験する

本稿では，基本的なシミュレーションの方法を，Verilog HDL言語を用いて説明します．テストベンチにスティミュラス記述とデザイン記述を配置し，シミュレーション結果を波形などで確認するという流れになります．シミュレータはModelSim Altera Web Edition 6.1g（米国Alteara社のOEMバージョン）を使用します．

● SRAM制御回路を検証する

今回用いるサンプル・コードはそれ自体が完全なデザインではなく，デザインの一部として用いられる可能性のあるSRAMと，SRAMへのデータ書き込みやSRAMからのデータ読み出しのシーケンスを制御するステート・マシンによって構成しています．このサンプル・コードはModelSimのインストール・ディレクトリに含まれるチュートリアルの題材に，若干の編集を加えたものです．

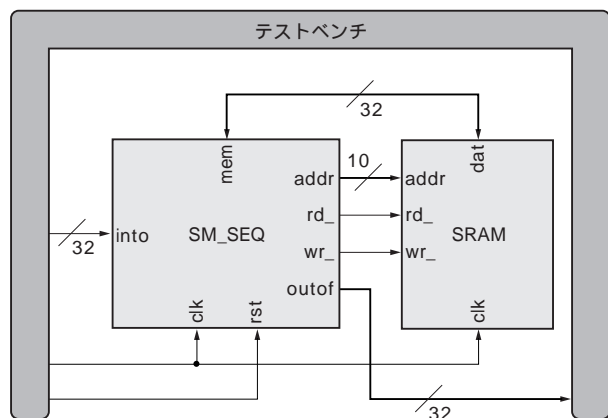



図2　SRAM制御回路とテストベンチの構成
回路は，SRAMと，SRAMへのデータ書き込みやSRAMからのデータ読み出しのシーケンスを制御するステート・マシンによって構成する．テストベンチでは，SRAM制御回路に対するスティミュラスを定義する．

テストベンチを含む全体の構成イメージを**図2**に示します．

テストベンチには，SRAMとステート・マシンを配置し，それに対するスティミュラスを定義します．テストベンチから検証対象のRTLコードへの制御は，clk信号とrst信号以外に，intoという32ビット幅のデータを用います．また，テストベンチからの観測はoutofという32ビット幅のデータを用います．

SRAMへの書き込みは，1ワード単位の書き込みと4ワード単位のブロック書き込みがあります．また，SRAMからの読み出しは1ワード単位の読み出しのみです．

32ビット幅を持つintoに対しては，

- SRAMの読み書き制御を行うオペレーション・コード（上位4ビットのみ）
- メモリのアドレス（下位10ビット）
- データそのもの（32ビット）

のいずれかが与えられます．

具体的な動作を説明します．オペレーション・コード2は，ワード単位の書き込みの指示になります．まず，intoの上位4ビットを0010に設定します．次のサイクルで書き込むべきアドレスを指定します．さらに，次のサイクルで書き込むデータを指定します．

オペレーション・コード3はブロック書き込みです．まず，intoの上位4ビットを0011に設定します．次のサイクルでブロック書き込みの開始アドレスを指定します．さらに，次のサイクルから4サイクルに渡り，書き込むべき四つのデータを指定します．ブロック書き込みの際は，開始アドレスとinca（increment-address）信号（ステート・マシンの内部信号）によってアドレスが自動的に1番地ずつ進むことになります．

テストベンチによる観測信号は，outofという32ビット・バスだけになります．outof信号に変化が発生するたびに，発生したシミュレーション時間とその時のoutofの値を表示します．

● SRAMモジュール

SRAMモジュールのソース・コードを**リスト1**に示します．

このモジュールは，32ビット×1024ワードが格納可能なSRAMです．datが32ビット・データの入出力ポート，addrが10ビットのアドレスです．rd_がアクティブ“L”のリード指定信号，wr_はアクティブ“L”のライト指定信号です．

リスト1　SRAMモジュールのソース・コード(beh_sram.v)

```
/* Simple Behavioral SRAM Model */
`timescale 1ns/100ps
module beh_sram(clk, dat, addr, rd_, wr_);

inout [31:0] dat;
input [9:0] addr;
input clk, rd_, wr_;

parameter M_DLY = 9;

reg  [31:0] mem [0:1023]; // memory array
reg  [31:0] dat_r;
tri [31:0] dat = rd_ ?  32'bZ : dat_r ;

initial begin
  dat_r = 0;
end

specify
  specparam ts = 9, th = 5, thr = 10;
  $setup(rd_, negedge clk, ts);
  $setup(wr_, negedge clk, ts);
  $setup(addr, negedge clk, ts);
  $hold(negedge clk, rd_, thr);
  $hold(negedge clk, wr_, th);
  $hold(negedge clk, addr, th);
endspecify

always @ (negedge clk)
  if (rd_ || wr_) begin
    if (!rd_)
      dat_r <= #M_DLY  mem[addr];
    if (!wr_)
      mem[addr] <= #M_DLY dat;
  end
  else begin
    if ((rd_ || wr_) == 0) begin
      $display($stime,, "Error: Simultaneous Reads &
                             Writes not supported.");
    end
  end

endmodule
```

リスト2　SM_SEQモジュールのソース・コード(sm_seq.v)

```
`timescale 1ns/100ps
module sm_seq ( into, outof, rst, clk, mem, addr, rd_,
                                                  wr_);

input [31:0] into;
output [31:0] outof;
input rst, clk;
inout [31:0] mem;
output [9:0] addr;
output rd_, wr_;

reg [31:0] in_reg, outof, w_data, r_data;
reg [9:0] addr;
reg wr_;
reg [7:0] ctrl;

tri [31:0]  mem = wr_ ? 32'bZ : w_data;

// instantiate the state machine module
sm sm_0( clk, rst, in_reg[31:28], a_wen_, wd_wen_,
                              rd_wen_, ctrl_wen_, inca);

wire rd_ = rd_wen_;
always @ (posedge clk)
  if (rst) begin
    in_reg <= 0; // get the input
    outof <= 0;  // send the output
    addr <= 0;
    w_data <= 0;
    wr_ <= 1'b1;
    r_data <= 0;
  end
  else begin
    in_reg <= into;   // get the input
    outof  <= r_data; // send the output

    if (!a_wen_)
      addr <= in_reg[9:0];
    else if (inca)
      addr <= addr + 1;

    if (!wd_wen_)
      w_data <= in_reg;

    wr_ <= wd_wen_;

    if (!rd_wen_)
      r_data <= mem;

    if (!ctrl_wen_)
      ctrl <= in_reg[7:0];
  end

endmodule
```

入出力ポートのdatに対しては，rd_が1'のとき，つまりリード状態でない場合に全ビットがハイ・インピーダンスとします．rd_が0のとき，つまりリード状態の場合に，指定されたアドレスのメモリ内容mem[addr]が，dat_rを通して読み出されます．`always`ブロックを見ると分かるように，リード時にはmem[addr]の内容がdat_rにアサインされ，ライト時にはdatの内容がmem[addr]に書き込まれます．もし，リードとライトが同時に指定された場合には，同時指定は許されないというエラー・メッセージを表示します．

このSRAMモジュールには，`specify`ブロックがあります．これは，リード，ライト，アドレスに対するセットアップ時間とホールド時間のチェックを行うタイミング情報です．`$setup`では，ts時間以前に値が確定することをチェックします．tsは`specparam`で9に設定しているので，ここではクロックの立ち下りに対して9ns以前に値が確定することをチェックすることになります．`$hold`は，クロックの立ち下りからth(＝5ns)またはthr(＝10ns)時間だけ値が変化しないことをチェックしています．

通常，RTL設計ではタイミング情報を記述する必要はあ

リスト3　SMモジュールのソース・コード(sm.v)

```
`timescale 1ns/100ps
module sm( clk, rst, opcode, a_wen_, wd_wen_, rd_wen_,
                                        ctrl_wen_, inca );

input clk, rst;
input [3:0] opcode;
output a_wen_, wd_wen_, rd_wen_, ctrl_wen_, inca;

parameter [10:0]     // state encodings
  IDLE      = 11'b00000000001,
  CTRL      = 11'b00000000010,
  WT_WD_1   = 11'b00000000100,
  WT_WD_2   = 11'b00000001000,
  WT_BLK_1  = 11'b00000010000,
  WT_BLK_2  = 11'b00000100000,
  WT_BLK_3  = 11'b00001000000,
  WT_BLK_4  = 11'b00010000000,
  WT_BLK_5  = 11'b00100000000,
  RD_WD_1   = 11'b01000000000,
  RD_WD_2   = 11'b10000000000;

reg [10:0] state, n_state;

// state machine output logic
wire a_wen_    = !( state[2] || state[4] || state[9]);
wire wd_wen_   = !( state[3] || state[5] || state[6] ||
                                    state[7] || state[8]);
wire rd_wen_   = !( state[10]);
wire inca      = ( state[6] || state[7] || state[8]);
wire ctrl_wen_ = !( state[1]);

// sequential logic
always @ (posedge clk or posedge rst)
  if (rst)
    state <= IDLE;
  else
    state <= n_state;

// next state logic
always @ (state or opcode)
  case (state)
    IDLE:       // IDLE
      case (opcode)
        0: // nop
            n_state = IDLE;
        1: // ctrl
            n_state = CTRL;
        2: // wt_wd
            n_state = WT_WD_1;
        3: // wt_blk
            n_state = WT_BLK_1;
        4: // rd_wd
            n_state = RD_WD_1;
        default: begin
            n_state = IDLE;
            $display ($time,,"illegal op received");
            end
      endcase
    CTRL:       // CTRL
        n_state = IDLE;
    WT_WD_1:        // WT_WD_1
        n_state = WT_WD_2;
    WT_WD_2:        // WT_WD_2
        n_state = IDLE;
    WT_BLK_1:       // WT_BLK_1
        n_state = WT_BLK_2;
    WT_BLK_2:       // WT_BLK_2
        n_state = WT_BLK_3;
    WT_BLK_3:       // WT_BLK_3
        n_state = WT_BLK_4;
    WT_BLK_4:       // WT_BLK_4
        n_state = WT_BLK_5;
    WT_BLK_5:       // WT_BLK_5
        n_state = IDLE;
    RD_WD_1:        // RD_WD_1
        n_state = RD_WD_2;
    RD_WD_2:        // RD_WD_2
        n_state = IDLE;
    default:
        n_state = IDLE;
    endcase

endmodule
```

りません．ただし，このようなRAMのモデルやASIC/FPGAベンダより提供される動作モデルやゲート・レベル・シミュレーション用のライブラリなどには，このような記述が含まれる場合があります．一度ファイルの内容を確認してみるとよいでしょう．

● SM_SEQモジュールとSMモジュール

SM_SEQモジュールのソース・コードを**リスト2**に示します．このモジュールは，ステート・マシンの下位モジュールSMを持つ階層構造になっています．SMモジュールのソース・コードを**リスト3**に示します．

入出力ポートmemによって，SRAMモジュールとのデータのやり取りを行います．SRAMのdatと同様に，wr_が‘1’ならば全ビットをハイ・インピーダンスとし，wr_が‘0’つまりライト状態の場合にはデータを書き込みます．

1）ステート・マシン

SMモジュールは，11種類の状態を持つステート・マシンです．**図3**にステート・ダイヤグラムを示します．

IDLE，CTRL，ワード単位の書き込みのWT_WD_1とWT_WD_2，ブロック単位の書き込みのWT_BLK_1～WT_BLK_5，ワード単位の読み出しのRD_WD_1とRD_WD_2から構成されます．ステートのエンコーディングには1ホットを用いています．

1ワード単位の書き込みでは，書き込むべきアドレスが指定されるサイクルがステートWT_WD_1，書き込むデータが指定されるサイクルがステートWT_WD_2となります．これらは一連のシーケンスを形成しなくてはならないため，ステート・マシンでは必ず次のステートとしてWT_WD_2を指定するように動かなくてはなりません．

ブロック単位の書き込みでは，書き込むべきブロックの開始アドレスを指定するサイクルがステートWT_BLK_1，書き込むべきデータが指定されるサイクルがステートWT_BLK_2，WT_BLK_3，WT_BLK_4，WT_BLK_5となります．WT_BLK_3からWT_BLK_5までの3サイク

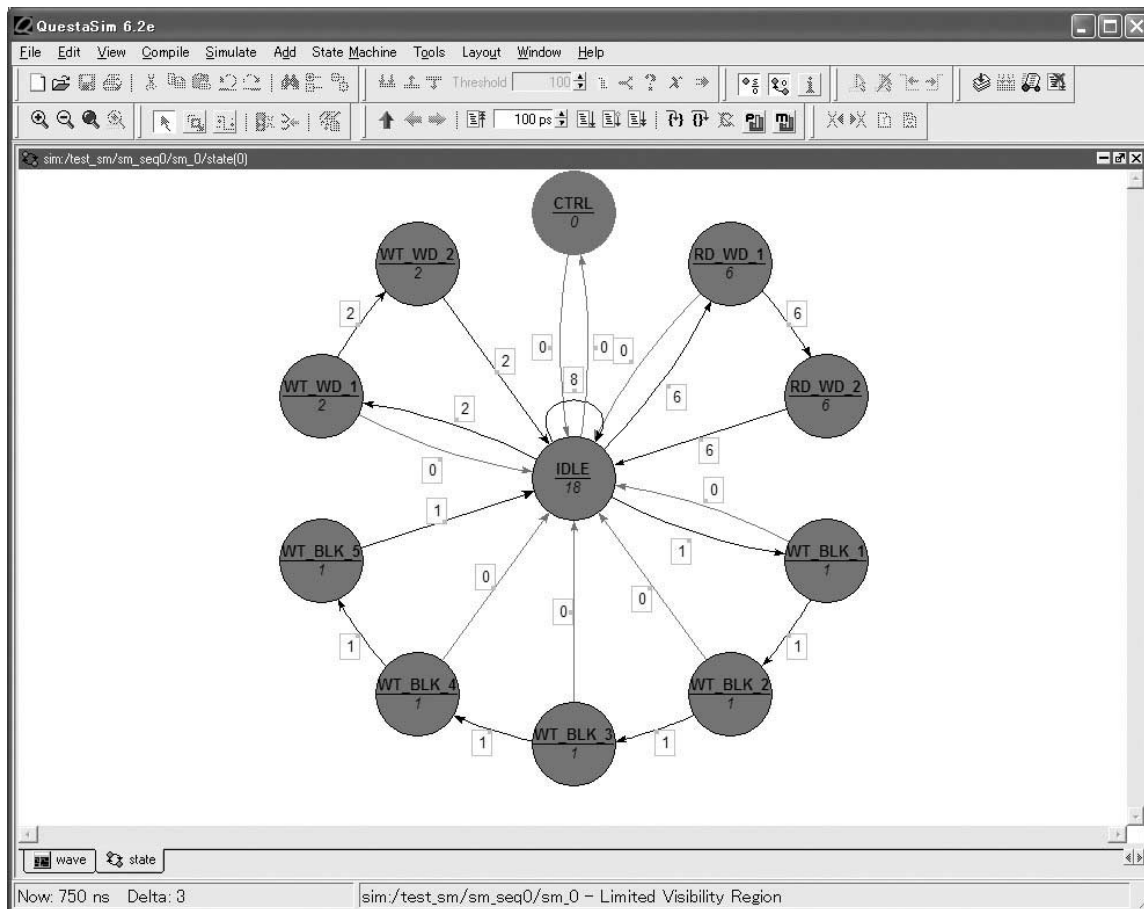



図3 SMモジュールのステート・ダイヤグラム
ModelSim/Questaのステート・マシン・カバレッジ機能を使用して表示したもの．OEMバージョン(AE/XE)にはカバレッジ機能はない．

ルの間はinca信号が1になることで，書き込むべきアドレスが自動的にインクリメントされるしくみです．

2）ステートによって決定される論理

a_wen_信号は，ステートWT_WD_1，WT_BLK_1，RD_WD_1など，各モードの1サイクル目のアクションを起こす際にアドレスの書き込みを許可する信号を構成します．この信号が“L”の場合には，SM_SEQモジュールにおいてin_reg[9:0]の10ビットをアドレスとして書き込みます．

wd_wen_信号は，信号WT_WD_2やWT_BLK_2～WT_BLK_5のように2サイクル目以降の書き込みを許可する信号です．“L”の場合にはSM_SEQモジュールにおいてin_regの内容を書き込み用のw_dataに割り当てることになります．

rd_wen_信号は，読み出しの許可信号です．メモリの内容をr_dataに割り当てます．

inca信号はブロック書き込みにおける3サイクル目から5サイクル目のアドレスの自動インクリメント用の信号です．

3）次のステート(n_state)を決定するための論理

次のステートは，現在のステートとopecodeによって決定されます．

現在のステートがIDLEの場合は，opecodeを参照し，それによってワード単位の書き込みや読み出し，ブロック単位の書き込みなどの制御モードに入ります．

現在のステートがWT_WD_1ステートの場合は，次のステートは必ずWT_WD_2でなくてはなりません．その次のステートはIDLEに戻らなくてはなりません．

ブロック書き込みでは，WT_BLK_1からWT_BLK_5へと順次ステートが進み，IDLEへと遷移しなくてはなりません．

ワード単位の読み出しの場合には，RD_WD_1→RD_WD_2→IDLEと遷移する必要があります．

● テストベンチの書き方

ステート・マシンのオペレーション・コードを検証するためのテストベンチを**リスト4**に示します．

1）`timescale

テストベンチの初めには，必ず `` `timescale ``のシミュレーション指示子でシミュレーションにおける基本単位と精度を記述します．**リスト4**では，シミュレーション単位を1ns，シミュレーション精度を100psとしています．シミュレーションにおける精度は，シミュレータの設定でも指定できます．しかし，テストベンチにも記述しておくことをおすすめします．`` `timescale ``は合成対象となるRTLには必要ありませんが（論理合成時には`#<数字>`で指定される時間や`$display`などのシステム・タスクは無視される），コンパイル順などの影響を受けてしまうため，RTLにも記述するほうが望ましいでしょう．

ModelSimではコンパイルされたソースをシミュレーションで読み込む（エラボレーション）時に`` `timescale ``の記述がないモジュールは，以下のようなワーニングが出力さ

リスト4　テストベンチのソース・コード（test_sm.v）

```
`timescale 1ns/100ps
module test_sm;

reg [31:0] into, outof;
reg rst, clk;
wire [31:0] out_wire, dat;
wire [9:0]  addr;

/* nop */
task nop;
  #5 into = {4'b0000,28'h0}; // op_word
endtask

/* the ctrl op */
task ctrl;
  input [7:0] data;
begin
  #5 into = {4'b0001,28'b0}; // ctrl_word
  @ (posedge clk)
  #5 into = data;
end
endtask

/* the wt_wd op */
task wt_wd;
  input [31:0] addr,data;
begin
  #5 into = {4'b0010,28'h0}; // op_word
  @ (posedge clk)
  #5 into = addr;
  @ (posedge clk)
  #5 into = data;
end
endtask

/* the wt_blk op */
task wt_blk;
  input [31:0] addr,data;
begin
  #5 into = {4'b0011,28'h0}; // op_word
  @ (posedge clk)
  #5 into = addr; // send address
  repeat (4)
    begin
      @ (posedge clk)
      #5 into = data; // send data
      data = data +1; // change the data word
    end
end
endtask

/* the rd_wd op */
task rd_wd;
  input [31:0] addr;
begin
  #5 into = {4'b0100,28'h0}; // op_word
  @ (posedge clk)
  #5 into = addr;
  @ (posedge clk)
  #5 into = 0;  // nop
end
endtask

/* illegal op */
task ill_op;
  #5 into = {4'b0101,28'h0};  // op word
endtask


initial
  into = 0;  // set to nop to start off

/* the clock */
initial
begin
  clk = 0;
  rst = 1;
  forever
    #10 clk = !clk;
end

always @(posedge clk)
  outof = #5 out_wire; // put output in register

always @ (outof)  // any change of outof
  $display ($time,,"outof = %h",outof);

/* tests */
initial
begin
  rst = 0;
  #5 rst = 1;
  #20 rst = 0;
  repeat(3) @ (posedge clk); // wait for 3 clocks
  repeat (40) begin
    @ (posedge clk) wt_wd('h10,'haa);
    @ (posedge clk) wt_wd('h20,'hbb);
    @ (posedge clk) wt_blk('h30,'hcc);
    @ (posedge clk) rd_wd('h10);
    @ (posedge clk) rd_wd('h20);
    @ (posedge clk) rd_wd('h30);
    @ (posedge clk) rd_wd('h31);
    @ (posedge clk) rd_wd('h32);
    @ (posedge clk) rd_wd('h33);
    @ (posedge clk) ill_op;
    @ (posedge clk) nop;
  end
  #100 $stop;
end

sm_seq  sm_seq0( into, out_wire, rst, clk, dat, addr,
                                                rd_, wr_);

beh_sram   sram_0(clk, dat, addr, rd_, wr_);

endmodule
```

れます．

** Warning: (vsim-3009) [TSCALE] - Module 'sm' does not have a `timescale directive in effect, but previous modules do.

ASICやFPGAのライブラリや提供されるIP（intellectual property）コアを使用する場合には，シミュレーション精度はそのライブラリに合わせるほうが望ましいでしょう．例えば，Altera社から提供されるライブラリのシミュレーション精度は1psとなっています（`` `timescale 1 ps/1 ps ``）．シミュレーション精度が違うと，ライブラリの動作が想定されていない動作になる場合があるので，注意が必要です．

2）task

テストベンチから検証対象のRTLコードに対しては，CLK，RST以外にはinto信号しか制御できません．そこで，この上位4ビットに対してオペレーション・コードを指定する処理を`task`を用いて実現しています．

`task`には`function`内では記述できない，#<数字>による時間指定や@によるイベント・トリガを含めることができます．テストベンチ内で，ある決まった処理を記述するには便利です．**リスト4**のテストベンチでは，NOP処理や制御，ワード単位の書き込み，ブロック単位の書き込み，ワード単位の読み出し，不正処理を作成しています．

3）initial

`initial`は，リセットからのシーケンスの記述です．はじめにリセットを入力した後，各`task`を呼び出して入力スティミュラスを与えています．スティミュラスはワード・データの書き込みが2回，ブロック書き込みが1回，ワード・データの読み込みが6回，不正なオペレーション・コード，NOPの順に与えられ，それらを40回繰り返します．

● シミュレーションの実行

シミュレーションのような繰り返しの処理が多い作業の場合，シミュレーションにおける一連の手順をスクリプトとして記述しておくほうが便利です．

1）スクリプトを使って実行する

ModelSimのシミュレーション・スクリプトを**リスト5**に示します．Windowsではこれをrun.batのようなファイル名（拡張子.bat）で保存しておくと，このファイルを

リスト5　シミュレーションのスクリプト（run.bat）

```
vlib work
vlog test_sm.v sm_seq.v sm.v beh_sram.v
vsim work.test_sm -do "add wave /*;run -all"
```

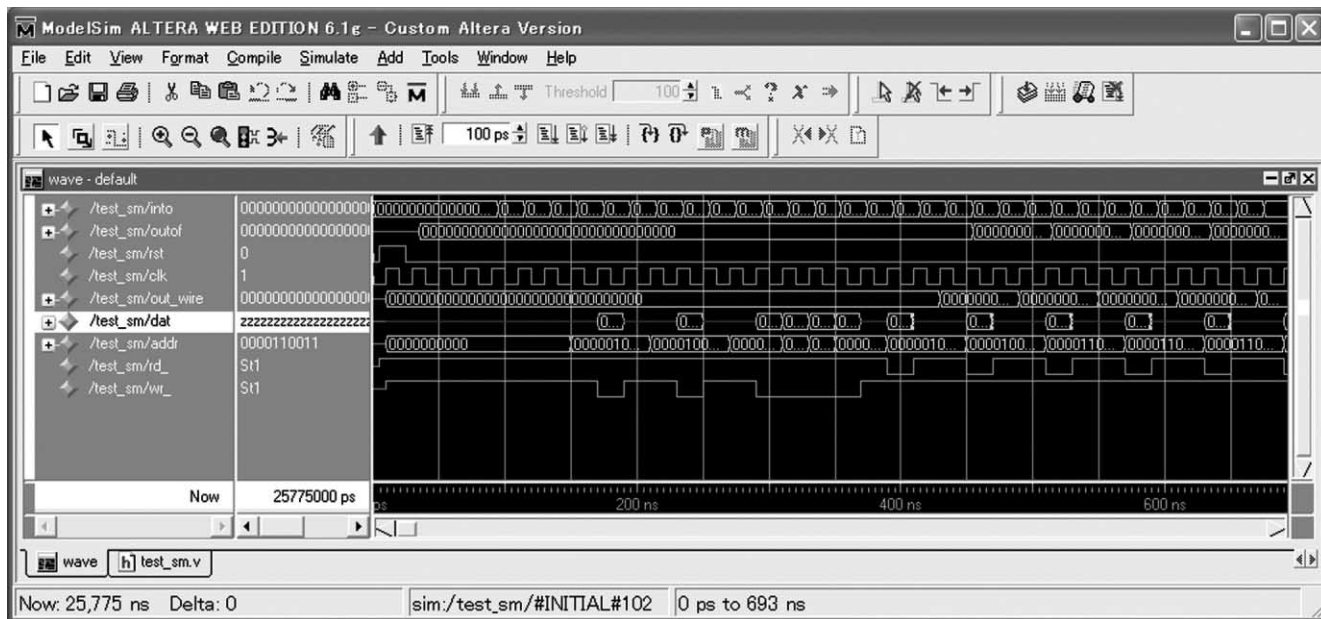



図4　シミュレーション結果

信号の値がバイナリ表示されている．「Simurate」→「Runtime Options」を選択し，DefaultsにあるDefault Radixを「Hexdecimal」などに変更すると表示形式を変更できる．

Windowsのエクスプローラなどからダブル・クリックするだけで実行可能になります（バッチ・ファイルで動作をさせるには，<Install Directory>\ModeltechAEweb6.1d\modelsim_ae\win32aloemへのパスを通しておく必要がある）．コンパイル（vlib，vlog）とシミュレーション用（vsim）のスクリプトは分けたほうが管理しやすい場合もあります．

シミュレーションを実行すると**図4**の波形が表示されます．

デフォルトの設定では，信号の値がバイナリ表示されています．そこで，「Simurate」→「Runtime Options」を選択し，Defaultsにある Default Radixを「Hexdecimal」などに変更すると，表示形式を変更できます．

リスト6　内部信号をシミュレーションするスクリプト（run_log.bat）

```
vlib work
vlog test_sm.v sm_seq.v sm.v beh_sram.v
vsim work.test_sm -do "log -r /*;run -all"
```

2）内部信号を観測する

ステート・マシンのstateレジスタを観測してみます．しかし，**リスト5**のスクリプトのままではstateの波形は表示できません．なぜなら，シミュレーション・コマンドのadd waveでは，最上位階層のすべての信号（/*）がwaveウィンドウに登録されるのですが，下位階層の信号は波形ログの対象とならないからです．

内部信号を観測するには，シミュレーションを開始する前に波形ログの対象としておく必要があります．シミュレーション・スクリプトを**リスト6**のように書き換えて再度実行します．

logコマンドは，信号をwaveウィンドウに登録せずに波形ログの対象とするコマンドです．-rオプションでは，最上位階層から下位階層までのすべての信号をログに保存する指定です．

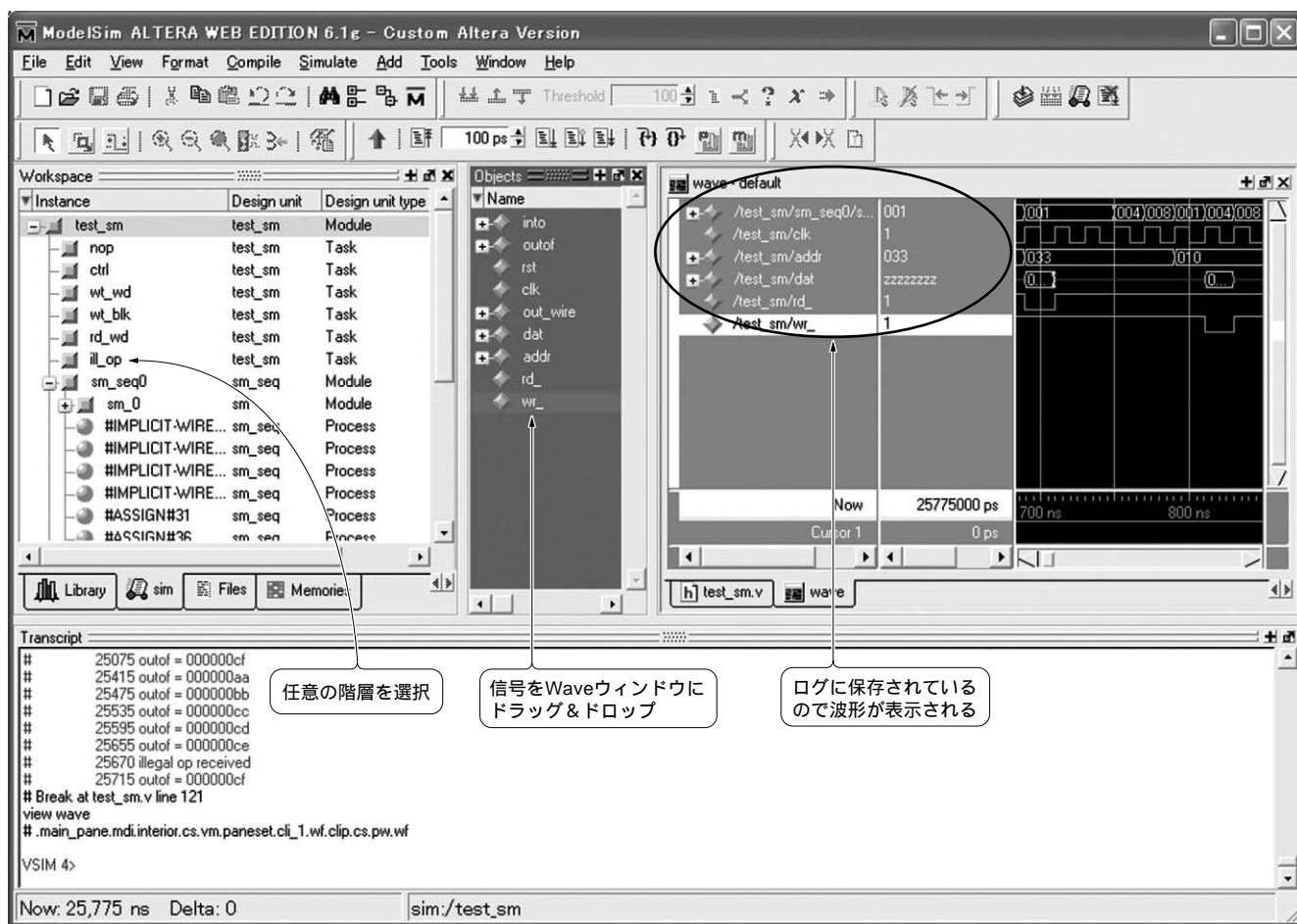



図5　内部信号の表示

波形ログの階層を選択し，Objectウィンドウから任意の信号をドラッグ＆ドロップする．

ModelSimのメニューから「View」→「Debug Windows」→「Wave」を選択すると，信号が登録されていないwaveウィンドウが表示されます．すべての信号は波形ログに登録されているので，階層を選択しObjectsウィンドウから任意の信号をドラッグ&ドロップすることで，波形を表示できます(**図5**)．

ただし，すべての信号を波形ログに保存する方法は便利ですが，登録する信号が多ければ多いほどシミュレーションの実行速度に影響します．そこで通常は，デバッグを行っている階層以下を波形ログに保存するほうがよいでしょう．

3）信号の状態を分かりやすく表示する

ステート・マシンのstate信号を波形で見ると，信号値で表示されるため，どのような状態かを判断しにくくなります．そこで，各ステートの値に対して意味のある名前を割り当てて表示させてみます．こうすることで，状態を瞬時に把握しやすくなります．

ModelSimでは，virtual typeというコマンドで信号値と文字列の対応を定義できます．**リスト7**のコマンドをModelSimのTranscriptウィンドウのコマンド・ラインから入力するか，一連のコマンドをファイルにしておいて，doコマンドで実行します．

VSIM > do virtual.do

これで**図6**のようにステートの状態をwaveウィンドウで表示できます．ステートがオペレーション・コードの値によって状態遷移しているのが確認できます．

virtual functionコマンドは，信号の値を比較する際にも用いることができます．datの値が32'h0000ccの時に1となる信号は，**リスト8**のように作成します．waveウィンドウには，**図7**のように新しいdat_ccの信号と波形が表示されます．

リスト7　信号値と文字列の対応を定義するスクリプト(virtual.do)

```
virtual type { ¥
 {0b00000000001 IDLE} ¥
 {0b00000000010 CTRL} ¥
 {0b00000000100 WT_WD_1} ¥
 {0b00000001000 WT_WD_2} ¥
 {0b00000010000 WT_BLK_1} ¥
 {0b00000100000 WT_BLK_2} ¥
 {0b00001000000 WT_BLK_3} ¥
 {0b00010000000 WT_BLK_4} ¥
 {0b00100000000 WT_BLK_5} ¥
 {0b01000000000 RD_WD_1} ¥
 {0b10000000000 RD_WD_2} ¥
 {default BAD_STATE}} myStateType
virtual function
  {(myStateType)/test_sm/sm_seq0/sm_0/state} myState
add wave /test_sm/sm_seq0/sm_0/myState
```

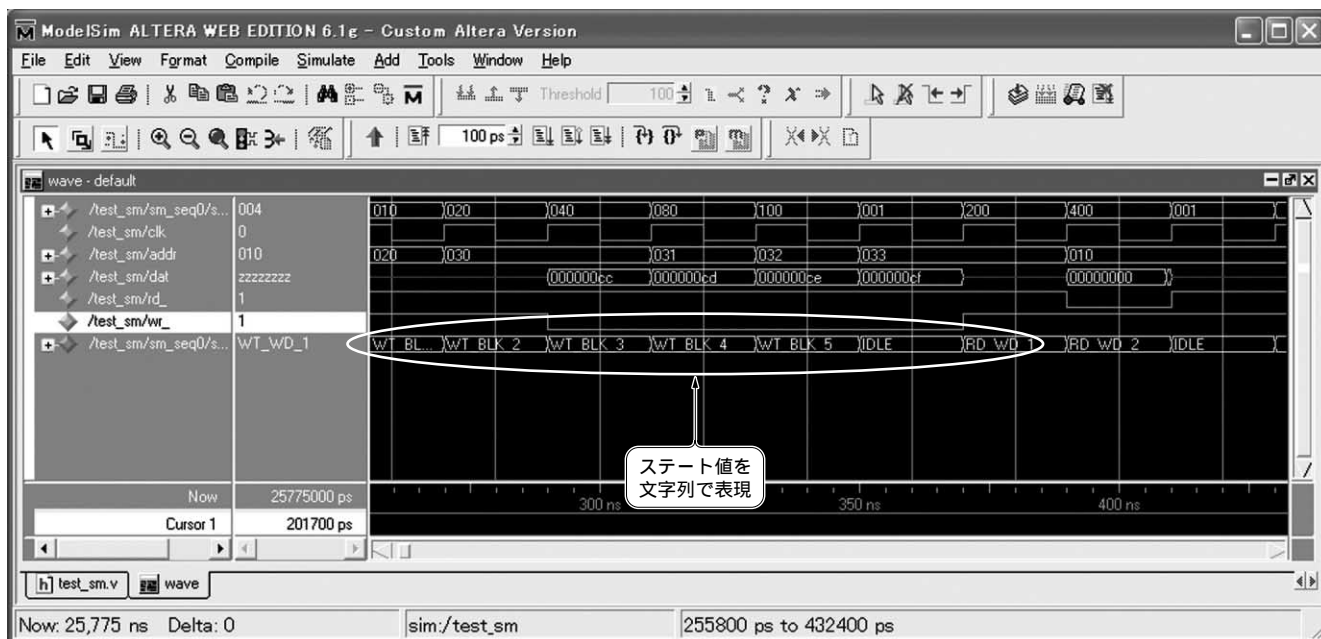



図6　ステートを分かりやすく表示

ステートが文字列で表示されていることが分かる．

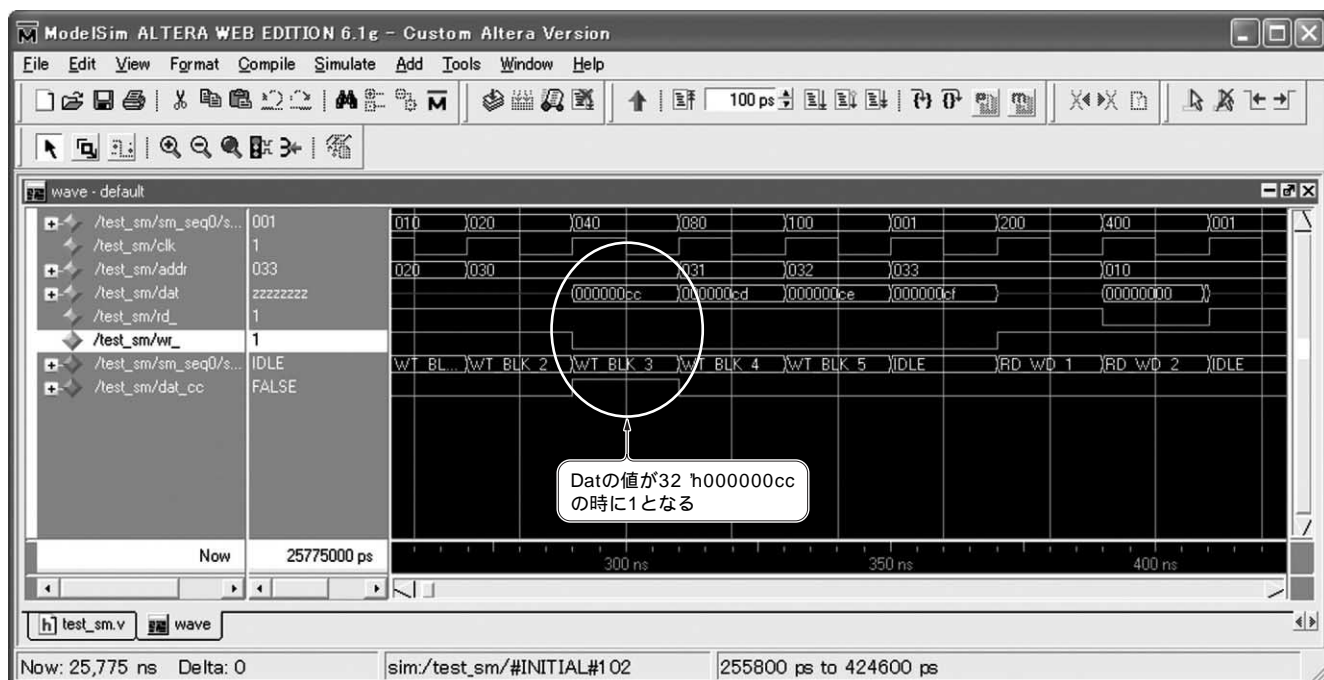



図7　信号の値を比較して表示

Datの値が32'h000000ccのときに1になっていることが分かる．

リスト8　信号の値を比較するスクリプト

```
virtual function { /test_sm/dat == 32'h0000cc } dat_cc
add wave /test_sm/dat_cc
```

waveウィンドウが選択された状態で「File」→「Save」選択すると，波形表示のフォーマットを保存できます．シミュレーションを終了すると，デフォルトでvsim.wlfという波形ログが保存されます．再度シミュレーションを行わずに波形だけを表示するには，ModelSimをViewモードで起動し，-doオプションで保存された波形フォーマットのファイルを指定すると，waveウィンドウに信号が登録された状態でModelSimが起動します注1．

vsim -view vsim.wlf -do wave.do

● まとめ

本稿により，テストベンチの基本的な記述方法とバッチファイルを作成したシミュレーションの実行を理解いただけたと思います．波形の表示や解析はデバッグの最も基本的な作業となるので，効果的なコマンドなどを使用して，効率良く検証/デバッグを行ってください．

みやじま・たけし
メンター・グラフィックス・ジャパン（株）

注1：ModelSimのOEMバージョン（AEやXE）は製品版のModelSim/Questaで作成したwlfを表示できない．OEMバージョンは自身で作成したwlfファイルのみ表示可能．

<筆者プロフィール>
宮島 健．検証，合成のほか，最近では高位合成のサポートをしている中年エンジニア．小学生の娘とスキーに行くのが趣味．最近忙しくて，あまり行けないのが悩み．